FINNEGAN, HENDERSON, FARABOW, GARRETT & DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, DC 20005-3315

202 • 408 • 4000
FACSIMILE 202 • 408 • 4400

ATLANTA
404•653•6400
PALO ALTO
650•849•6600

WRITER'S DIRECT DIAL NUMBER:

(202) 408-4470

TOKYO
011•813•3431•6943
BRUSSELS
011•322•646•0353

December 10, 1999

ATTORNEY DOCKET NO. 6502.0287

Box PATENT APPLICATION
Assistant Commissioner for Patents
Washington, D.C. 20231

Re: New U.S. Patent Application
Title: CHANNEL-SPECIFIC FILE SYSTEM VIEWS IN A PRIVATE NETWORK USING A PUBLIC-NETWORK INFRASTRUCTURE
Inventors: Germano CARONNI, Amit GUPTA, Tom R. MARKSON, Sandeep KUMAR, Christoph L. SCHUBA, and Glen C. SCOTT

Sir:

We enclose the following papers for filing in the United States Patent and Trademark Office in connection with the above patent application.

1. Application - 33 pages, including 6 independent claims and 20 claims total.

2. Drawings - 13 sheets of informal drawings (Figures 1-12).

This application is being filed under the provisions of 37 C.F.R. § 1.53(b) and § 1.53(f). Applicants await notification from the Patent and Trademark Office of the time set for filing the Declaration.

Please accord this application a serial number and filing date.

No filing fee is being paid at this time. With the exception of the filing fee, the Commissioner is hereby authorized to charge any fees due, including fees under 37 C.F.R. § 1.16 or § 1.17, during the pendency of this application to our Deposit Account No. 06-0916.

Respectfully submitted,

FINNEGAN, HENDERSON, FARABOW, GARRETT & DUNNER, L.L.P.

By: _____
Michael L. Kiklis
Reg. No. 38,939

Sun's Ref. No. P4310
Attorney Docket No.: 06502.0287

UNITED STATES PATENT APPLICATION

OF

GERMANO CARONNI

AMIT GUPTA

SANDEEP KUMAR

TOM R. MARKSON

CHRISTOPH L. SCHUBA

GLENN C. SCOTT

FOR

CHANNEL-SPECIFIC FILE SYSTEM VIEWS IN A PRIVATE NETWORK USING A PUBLIC-NETWORK INFRASTRUCTURE

RELATED APPLICATIONS

The following identified U.S. patent applications are relied upon and are incorporated by reference in this application.

U.S. Patent Application No. _____, entitled "SYSTEM AND METHOD FOR SEPARATING ADDRESSES FROM THE DELIVERY SCHEME IN A VIRTUAL PRIVATE NETWORK," bearing attorney docket no. 06502.0280, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "TRULY ANONYMOUS COMMUNICATIONS USING SUPERNETS WITH THE PROVISION OF TOPOLOGY HIDING," bearing attorney docket no. 06502.0281, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "METHOD AND SYSTEM FOR FACILITATING RELOCATION OF DEVICES ON A NETWORK," bearing attorney docket no. 06502.0283, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "SANDBOXING APPLICATIONS IN A PRIVATE NETWORK USING A PUBLIC-NETWORK INFRASTRUCTURE," bearing attorney docket no. 06502.0284, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "SECURE ADDRESS RESOLUTION FOR A PRIVATE NETWORK USING A PUBLIC NETWORK INFRASTRUCTURE," bearing attorney docket no. 06502.0285, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "DECOUPLING ACCESS CONTROL FROM KEY MANAGEMENT IN A NETWORK," bearing attorney docket no. 06502.0286, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "PRIVATE NETWORK USING A PUBLIC-NETWORK INFRASTRUCTURE," bearing attorney docket no. 06502.0288, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "SYSTEM AND METHOD FOR ENABLING SCALABLE SECURITY IN A VIRTUAL PRIVATE NETWORK," bearing attorney docket no. 06502.0289, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "USING MULTICASTING TO PROVIDE ETHERNET-LIKE COMMUNICATION BEHAVIOR TO SELECTED PEERS ON A NETWORK," bearing attorney docket no. 06502.0290, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "ANYCASTING IN A PRIVATE NETWORK USING A PUBLIC NETWORK INFRASTRUCTURE," bearing attorney docket no. 06502.0291, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "SCALABLE SECURITY ASSOCIATIONS FOR GROUPS FOR USE IN A PRIVATE NETWORK USING A PUBLIC-NETWORK INFRASTRUCTURE," bearing attorney docket no. 06502.0292, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "ENABLING SIMULTANEOUS PROVISION OF INFRASTRUCTURE SERVICES," bearing attorney docket no. 06502.0293, and filed on the same date herewith.

FIELD OF THE INVENTION

The present invention relates generally to data processing systems and, more particularly, to providing channel-specific file system views in a private network using a public-network infrastructure.

BACKGROUND OF THE INVENTION

As part of their day-to-day business, many organizations require an enterprise network, a private network with lease lines, dedicated channels, and network connectivity devices, such as routers, switches, and bridges. These components, collectively known as the network's "infrastructure," are very expensive and require a staff of information technology personnel to maintain them. This maintenance requirement is burdensome on many organizations whose main business is not related to the data processing industry (e.g., a clothing manufacturer) because they are not well suited to handle such data processing needs.

Another drawback to enterprise networks is that they are geographically restrictive. The term "geographically restrictive" refers to the requirement that if a user is not physically located such that they can plug their device directly into the enterprise network, the user cannot typically utilize it. To alleviate the problem of geographic restrictiveness, virtual private networks have been developed.

In a virtual private network (VPN), a remote device or network connected to the Internet may connect to the enterprise network through a firewall. This allows the remote device to access resources on the enterprise network even though it may not be located near any component of the enterprise network. For example, Fig. 1 depicts a VPN 100, where enterprise network 102 is connected to the Internet 104 via firewall 106. By using VPN 100, a remote device D_1 108 may

communicate with enterprise network 102 via Internet 104 and firewall 106. Thus, D_1 108 may be plugged into an Internet portal virtually anywhere within the world and make use of the resources on enterprise network 102.

To perform this functionality, D_1 108 utilizes a technique known as tunneling to ensure that the communication between itself and enterprise network 102 is secure in that it cannot be viewed by an interloper. "Tunneling" refers to encapsulating one packet inside another when packets are transferred between end points (e.g., D_1 108 and VPN software 109 running on firewall 106). The packets may be encrypted at their origin and decrypted at their destination. For example, Fig. 2A depicts a packet 200 with a source Internet protocol (IP) address 202, a destination IP address 204, and data 206. It should be appreciated that packet 200 contains other information not depicted, such as the source and destination port. As shown in Fig. 2B, the tunneling technique forms a new packet 208 out of packet 200 by encrypting it and adding both a new source IP address 210 and a new destination IP address 212. In this manner, the contents of the original packet (i.e., 202, 204, and 206) are not visible to any entity other than the destination. Referring back to Fig. 1, by using tunneling, remote device D_1 108 may communicate and utilize the resources of the enterprise network 102 in a secure manner.

Although VPNs alleviate the problem of geographic restrictiveness, they impose significant processing overhead when two remote devices communicate. For example, if remote device D_1 108 wants to communicate with remote device D_2 110, D_1 sends a packet using tunneling to VPN software 109, where the packet is decrypted and then transferred to the enterprise network 102. Then, the enterprise network 102 sends the packet to VPN software 109, where it is encrypted again and transferred to D_2. Given this processing overhead, it is burdensome for two remote devices to

communicate in a VPN environment. It is therefore desirable to alleviate the need of organizations to maintain their own network infrastructure as well as to improve communication between remote devices.

SUMMARY OF THE INVENTION

Methods and systems consistent with the present invention provide a private network that uses components from a public-network infrastructure. Nodes of the private network can be located on virtually any device in the public network (e.g., the Internet), and both their communication and utilization of resources occur in a secure manner. As a result, the users of this private network benefit from their network infrastructure being maintained for them as part of the public-network infrastructure, while the level of security they receive is similar to or even stronger than that provided by conventional private networks. Additionally, the nodes of the private network are not geographically restricted in that they can be connected to the private network from virtually any portal to the Internet in the world.

This private network provides for channel-specific file system views such that nodes on one channel typically have a different view of the file system than nodes on a different channel. This allows for a virtual partitioning of the entire network, including the storage devices on the network, into separate partitions for each channel. Such functionality is beneficial to large organizations that want to restrict access to various information.

In accordance with systems consistent with the present invention a distributed system with a network of devices is provided. The devices have nodes communicating over a first channel and a second channel, and one of the devices comprises a secondary storage device, a memory, and a

processor. The secondary storage device comprises of a plurality of file system entities, where nodes communicating over the first channel may access the a first of the file system entities, and the nodes communicating over the second channel may access a second of the file system entities. The memory contains an operating system that restricts access to the first file system entity to the nodes communicating over the first channel and that restricts access to the second file system entity to the nodes communicating over the second channel. The memory also contains one of the nodes that communicates over the first channel and that sends a request to access the first file system entity to the operating system. The processor runs the one node and the operating system.

In another implementation, a method is provided in a distributed system with a network of nodes communicating over channels. This method receives a request from one of the nodes to access a file system entity, where the file system entity has an associated authorized one of the channels. This method also determines whether a node communicates over the authorized channel and accesses the file system entity when it has determined that the node communicates over the authorized channel.

In a yet another implementation, a method is provided in a distributed system with a network of nodes communicating over channels. The network has a file system with a plurality of file system entities, each with an associated authorized channel. This method receives a request from one of the nodes communicating over one of the channels to view the file system, determines the file system entities that have an authorized channel as the one channel, and returns an indication of the determined file system entities.

In another implementation consistent with the present invention, a computer-readable memory device encoded with a data structure is provided for use by an operating system that

provides channel-specific file views of a file system. The data structure has an entry for an indication of a file system entity. Additionally, the data structure has an indication of a channel that is used by the operating system to restrict access to the file system entity to nodes that communicate over the indicated channel.

BRIEF DESCRIPTION OF THE DRAWINGS

This invention is pointed out with particularity in the appended claims. The above and further advantages of this invention may be better understood by referring to the following description taken in conjunction with the accompanying drawings, in which:

Fig. 1 depicts a conventional virtual private network (VPN) system;

Fig. 2A depicts a conventional network packet;

Fig. 2B depicts the packet of Fig. 2A after it has been encrypted in accordance with a conventional tunneling technique;

Fig. 3 depicts a data processing system suitable for use with methods and systems consistent with the present invention;

Fig. 4 depicts the nodes depicted in Fig. 3 communicating over multiple channels;

Fig. 5 depicts two devices depicted in Fig. 3 in greater detail;

Figs. 6A and 6B depict a flow chart of the steps performed when a node joins a VPN in a manner consistent with the present invention;

Fig. 7 depicts a flow chart of the steps performed when sending a packet from a node of the VPN in a manner consistent with the present invention;

Fig. 8 depicts a flow chart of the steps performed when receiving a packet by a node of the VPN in a manner consistent with the present invention;

Fig. 9 depicts a flow chart of the steps performed when logging out of a VPN in a manner consistent with the present invention;

Fig. 10 depicts a file system view table used for providing channel-specific file system views in a manner consistent with the present invention;

Fig. 11 depicts a flow chart of the steps performed by the NameI function depicted in Fig. 5; and

Fig. 12 depicts a flow chart of the steps performed by the GetDir function depicted in Fig. 5.

DETAILED DESCRIPTION

Methods and systems consistent with the present invention provide a "Supernet," which is a private network that uses components from a public-network infrastructure. A Supernet allows an organization to utilize a public-network infrastructure for its enterprise network so that the organization no longer has to maintain a private network infrastructure; instead, the organization may have the infrastructure maintained for them by one or more service providers or other organizations that specialize in such connectivity matters. As such, the burden of maintaining an enterprise network is greatly reduced. Moreover, a Supernet is not geographically restrictive, so a user may plug their device into the Internet from virtually any portal in the world and still be able to use the resources of their private network in a secure and robust manner.

The Supernet provides channel-specific file system views such that the nodes on one channel typically have a different view of the network file system than nodes on a different channel. Such channel-specific views are enforced by the operating system on each device that has a node running.

Overview

Fig. 3 depicts a data processing system 300 suitable for use with methods and systems consistent with the present invention. Data processing system 300 comprises a number of devices, such as computers 302-312, connected to a public network, such as the Internet 314. A Supernet's infrastructure uses components from the Internet because devices 302, 304, and 312 contain nodes that together form a Supernet and that communicate by using the infrastructure of the Internet. These nodes 316, 318, 320, and 322 are communicative entities (e.g., processes) running within a particular device and are able to communicate among themselves as well as access the resources of the Supernet in a secure manner. When communicating among themselves, the nodes 316, 318, 320, and 322 serve as end points for the communications, and no other processes or devices that are not part of the Supernet are able to communicate with the Supernet's nodes or utilize the Supernet's resources. The Supernet also includes an administrative node 306 to administer to the needs of the Supernet. It should be noted that since the nodes of the Supernet rely on the Internet for connectivity, if the device on which a node is running relocates to another geographic location, the device can be plugged into an Internet portal and the node running on that device can quickly resume the use of the resources of the Supernet. It should also be noted that since a Supernet is layered on top of an existing network, it operates independently of the transport layer. Thus, the nodes of a

Supernet may communicate over different transports, such as IP, IPX, X.25, or ATM, as well as different physical layers, such as RF communication, cellular communication, satellite links, or land-based links.

As shown in Fig. 4, a Supernet includes a number of channels that its nodes 316-322 can communicate over. A "channel" refers to a collection of virtual links through the public-network infrastructure that connect the nodes on the channel such that only these nodes can communicate over it. A node on a channel may send a message to another node on that channel, known as a unicast message, or it can send a message to all other nodes on that channel, known as a multicast message. For example, channel 1 402 connects node A 316 and node C 320, and channel 2 404 connects node B 318, node C 320, and node D 322. Each Supernet has any number of preconfigured channels over which the nodes on that channel can communicate. In an alternative embodiment, the channels are dynamically defined.

In addition to communication, the channels may be used to share resources. For example, channel 1 402 may be configured to share a file system as part of node C 320 such that node A 316 can utilize the file system of node C in a secure manner. In this case, node C 320 serves as a file system manager by receiving file system requests (e.g., open, close, read, write, etc.) and by satisfying the requests by manipulating a portion of the secondary storage on its local machine. To maintain security, node C 320 stores the data in an encrypted form so that it is unreadable by others. Such security is important because the secondary storage may not be under the control of the owners of the Supernet, but may instead be leased from a service provider. Additionally, channel 2 404 may be configured to share the computing resources of node D 322 such that nodes B 318 and C 320 send

code to node D for execution. By using channels in this manner, resources on a public network can be shared in a secure manner.

A Supernet provides a number of features to ensure secure and robust communication among its nodes. First, the system provides authentication and admission control so that nodes become members of the Supernet under strict control to prevent unauthorized access. Second, the Supernet provides communication security services so that the sender of a message is authenticated and communication between end points occurs in a secure manner by using encryption. Third, the system provides key management to reduce the possibility of an intruder obtaining an encryption key and penetrating a secure communication session. The system does so by providing one key per channel and by changing the key for a channel whenever a node joins or leaves the channel. Alternatively, the system may use a different security policy.

Fourth, the system provides address translation in a transparent manner. Since the Supernet is a private network constructed from the infrastructure of another network, the Supernet has its own internal addressing scheme, separate from the addressing scheme of the underlying public network. Thus, when a packet from a Supernet node is sent to another Supernet node, it travels through the public network. To do so, the Supernet performs address translation from the internal addressing scheme to the public addressing scheme and vice versa. To reduce the complexity of Supernet nodes, system-level components of the Supernet perform this translation on behalf of the individual nodes so that it is transparent to the nodes. Another benefit of the Supernet's addressing is that it uses an IP-based internal addressing scheme so that preexisting programs require little modification to run within a Supernet.

Fifth, the Supernet provides operating system-level enforcement of node compartmentalization in that an operating system-level component treats a Supernet node running on a device differently than it treats other processes on that device. This component (i.e., a security layer in a protocol stack) recognizes that a Supernet node is part of a Supernet, and therefore, it enforces that all communications to and from this node travel through the security infrastructure of the Supernet such that this node can communicate with other members of the Supernet and that non-members of the Supernet cannot access this node. Additionally, this operating system-level enforcement of node compartmentalization allows more than one Supernet node to run on the same machine, regardless of whether the nodes are from the same Supernet, and allows nodes of other networks to run on the same machine as a Supernet node.

Lastly, the Supernet allows channel-specific file system views. That is, file system entities, such as directories and files, are associated with a channel, and access to a file system entity is restricted to nodes communicating over the channel associated with that file system entity. In this manner, the file system on the entire Supernet can provide different views for different channels, thus facilitating internal network security. For example, in an organization, the finance department may have access to certain file system entities that the rest of the organization should not. By using channel-specific file system views, such a partitioning is easily achieved.

Implementation Details

Fig. 5 depicts administrative machine 306 and device 302 in greater detail, although the other devices 304 and 308-312 may contain similar components. Device 302 and administrative machine 306 communicate via Internet 314. Each device contains similar components, including a memory

502, 504; secondary storage 506, 508; a central processing unit (CPU) 510, 512; an input device 514, 516; and a video display 518, 520. One skilled in the art will appreciate that these devices may contain additional or different components. Memory 504 of administrative machine 306 includes the SASD process 540, VARPD 548, and KMS 550 all running in user mode. That is, CPU 512 is capable of running in at least two modes: user mode and kernel mode. When CPU 512 executes programs running in user mode, it prevents them from directly manipulating the hardware components, such as video display 518. On the other hand, when CPU 512 executes programs running in kernel mode, it allows them to manipulate the hardware components. Memory 504 also contains a VARPDB 551 and a TCP/IP protocol stack 552 that are executed by CPU 512 running in kernel mode. TCP/IP protocol stack 552 contains a TCP/UDP layer 554 and an IP layer 556, both of which are standard layers well known to those of ordinary skill in the art. Secondary storage 508 contains a configuration file 558 that stores various configuration-related information (described below) for use by SASD 540.

SASD 540 represents a Supernet: there is one instance of an SASD per Supernet, and it both authenticates nodes and authorizes nodes to join the Supernet. VARPD 548 has an associated component, VARPDB 551, into which it stores mappings of the internal Supernet addresses, known as a node IDs, to the network addresses recognized by the public-network infrastructure, known as the real addresses. The "node ID" may include the following: a Supernet ID (e.g., 0x123), reflecting a unique identifier of the Supernet, and a virtual address, comprising an IP address (e.g., 10.0.0.1). The "real address" is an IP address (e.g., 10.0.0.2) that is globally unique and meaningful to the public-network infrastructure. In a Supernet, one VARPD runs on each machine, and it may play two roles. First, a VARPD may act as a server by storing all address mappings for a particular

Supernet into its associated VARPDB. Second, regardless of its role as a server or not, each VARPD assists in address translation for the nodes on its machine. In this role, the VARPD stores into its associated VARPDB the address mappings for its nodes, and if it needs a mapping that it does not have, it will contact the VARPD that acts as the server for the given Supernet to obtain it.

KMS 550 performs key management by generating a new key every time a node joins a channel and by generating a new key every time a node leaves a channel. There is one KMS per channel in a Supernet.

To configure a Supernet, a system administrator creates a configuration file 558 that is used by SASD 540 when starting or reconfiguring a Supernet. This file may specify: (1) the Supernet name, (2) all of the channels in the Supernet, (3) the nodes that communicate over each channel, (4) the address of the KMS for each channel, (5) the address of the VARPD that acts as the server for the Supernet, (6) the user IDs of the users who are authorized to create Supernet nodes, (7) the authentication mechanism to use for each user of each channel, and (8) the encryption algorithm to use for each channel. Although the configuration information is described as being stored in a configuration file, one skilled in the art will appreciate that this information may be retrieved from other sources, such as databases or interactive configurations.

After the configuration file is created, it is used to start a Supernet. For example, when starting a Supernet, the system administrator first starts SASD, which reads the configuration information stored in the configuration file. Then, the administrator starts the VARPD on the administrator's machine, indicating that it will act as the server for the Supernet and also starts the KMS process. After this processing has completed, the Supernet is ready for nodes to join it.

Memory 502 of device 302 contains SNlogin script 522, SNlogout script 524, VARPD 526, KMC 528, KMD 530, and node A 522, all running in user mode. Memory 502 also includes TCP/IP protocol stack 534 and VARPDB 536 running in kernel mode. In addition, memory 502 includes operating system 560 running in kernel mode, which further includes three components: NameI function 560, GetDir function 562, and file system view table (FSVT) 564. An operating system suitable for use with the present invention includes the Unix operating system, although one skilled in the art will appreciate that the present invention may work with other operating systems as well.

SNlogin 522 is a script used for logging into a Supernet. Successfully executing this script results in a Unix shell from which programs (e.g., node A 522) can be started to run within the Supernet context, such that address translation and security encapsulation is performed transparently for them and all they can typically access is other nodes on the Supernet. Alternatively, a parameter may be passed into SNlogin 522 that indicates a particular process to be automatically run in a Supernet context. Once a program is running in a Supernet context, all programs spawned by that program also run in the Supernet context, unless explicitly stated otherwise. SNlogout 524 is a script used for logging out of a Supernet. Although both SNlogin 522 and SNlogout 524 are described as being scripts, one skilled in the art will appreciate that their processing may be performed by another form of software. VARPD 526 performs address translation between node IDs and real addresses. KMC 528 is the key management component for each node that receives updates whenever the key for a channel ("the channel key") changes. There is one KMC per node per channel. KMD 530 receives requests from SNSL 542 of the TCP/IP protocol stack 534 when a packet is received and accesses the appropriate KMC for the destination node to retrieve the appropriate key to decrypt the packet. Node A 532 is a Supernet node running in a Supernet context.

TCP/IP protocol stack 534 contains a standard TCP/UDP layer 538, two standard IP layers (an inner IP layer 540 and an outer IP layer 544), and a Supernet security layer (SNSL) 542, acting as the conduit for all Supernet communications. To conserve memory, both inner IP layer 540 and outer IP layer 544 may share the same instance of the code of an IP layer. SNSL 542 performs security functionality as well as address translation. It also caches the most recently used channel keys for ten seconds. Thus, when a channel key is needed, SNSL 542 checks its cache first, and if it is not found, it requests KMD 530 to contact the appropriate KMC to retrieve the appropriate channel key. Two IP layers 540, 544 are used in the TCP/IP protocol stack 534 because both the internal addressing scheme and the external addressing scheme are IP-based. Thus, for example, when a packet is sent, inner IP layer 540 receives the packet from TCP/UDP layer 538 and processes the packet with its node ID address before passing it to the SNSL layer 542, which encrypts it, prepends the real source IP address and the real destination IP address, and then passes the encrypted packet to outer IP layer 544 for sending to the destination.

SNSL 542 utilizes VARPDB 536 to perform address translation. VARPDB stores all of the address mappings encountered thus far by SNSL 542. If SNSL 542 requests a mapping that VARPDB 536 does not have, VARPDB communicates with the VARPD 526 on the local machine to obtain the mapping. VARPD 526 will then contact the VARPD that acts as the server for this particular Supernet to obtain it.

The Supernet uses the OS 558 to provide channel-specific file system views by using the NameI function 560 and the GetDir function 560. NameI function 560 provides a requesting node with access to a file system entity in file system 566 by accessing the file system view table 564 to determine if the requesting node communicates over the appropriate channel to be authorized to

access that file system entity. GetDir function 562 receives a request from a node to view that portion of file system 566 that it is entitled to view. To respond to this request, GetDir function 562 accesses file system view table 564 to identify the file system entities that the node is authorized to access and returns a list of those file system entities to the node. Although described relative to file system 566, it should be appreciated that the file system view table provides mappings to all the storage devices on the Supernet. Thus, both the NameI function and the GetDir function operate on the file system entities located anywhere within the Supernet. As such, the file system, as used in herein, refers to the file system of the Supernet, not any one particular device.

Although aspects of the present invention are described as being stored in memory, one skilled in the art will appreciate that these aspects can also be stored on or read from other types of computer-readable media, such as secondary storage devices, like hard disks, floppy disks, or CD-ROM; a carrier wave from a network, such as the Internet; or other forms of RAM or ROM either currently known or later developed. Additionally, although a number of the software components are described as being located on the same machine, one skilled in the art will appreciate that these components may be distributed over a number of machines.

Figs. 6A and 6B depict a flow chart of the steps performed when a node joins a Supernet. The first step performed is that the user invokes the SNlogin script and enters the Supernet name, their user ID, their password, and a requested virtual address (step 602). Of course, this information depends on the particular authentication mechanism used. Upon receiving this information, the SNlogin script performs a handshaking with SASD to authenticate this information. In this step, the user may request a particular virtual address to be used, or alternatively, the SASD may select one for them. Next, if any of the information in step 602 is not validated by SASD (step 604), processing

ends. Otherwise, upon successful authentication, SASD creates an address mapping between a node ID and the real address (step 606). In this step, SASD concatenates the Supernet ID with the virtual address to create the node ID, obtains the real address of the SNlogin script by querying network services in a well-known manner, and then registers this information with the VARPD that acts as the server for this Supernet. This VARPD is identified in the configuration file.

After creating the address mapping, SASD sends a copy of the master version of the file system view table 564 to the operating system 558 resident in the kernel space of the device of the node requesting to join the Supernet (step 607). The file system view table along with the operating system provide channel-specific file system view as discussed below.

Subsequently, SASD informs the KMS that there is a new Supernet member that has been authenticated and admitted (step 608). In this step, SASD sends the node ID and the real address to KMS who then generates a key ID, a key for use in communicating between the node's KMC and the KMS ("a node key"), and updates the channel key for use in encrypting traffic on this particular channel (step 610). Additionally, KMS sends the key ID and the node key to SASD and distributes the channel key to all KMCs on the channel as a new key because a node has just been added to the channel. SASD receives the key ID and the node key from KMS and returns it to SNlogin (step 612). After receiving the key ID and the node key from SASD, SNlogin starts a KMC for this node and transmits to the KMC the node ID, the key ID, the node key, the address of the VARPD that acts as the server for this Supernet, and the address of KMS (step 614). The KMC then registers with the KMD indicating the node it is associated with, and KMC registers with KMS for key updates (step 616). When registering with KMS, KMC provides its address so that it can receive updates to the channel key via the Versakey protocol. The Versakey protocol is described in greater detail

in IEEE Journal on Selected Areas in Communication, Vol. 17, No. 9, 1999, pp. 1614-1631. After registration, the KMC will receive key updates whenever a channel key changes on one of the channels that the node communicates over.

Next, SNlogin configures SNSL (step 618 in Fig. 6B). In this step, SNlogin indicates which encryption algorithm to use for this channel and which authentication algorithm to use, both of which are received from the configuration file via SASD. SNSL stores this information in an access control list. In accordance with methods and systems consistent with present invention, any of a number of well-known encryption algorithms may be used, including the Data Encryption Standard (DES), Triple-DES, the International Data Encryption Algorithm (IDEA), and the Advanced Encryption Standard (AES). Also, RC2, RC4, and RC5 from RSA Incorporated may be used as well as Blowfish from Counterpane.com. Additionally, in accordance with methods and systems consistent with the present invention, any of a number of well-known authentication algorithms may be used, including Digital Signatures, Kerberos, Secure Socket Layer (SSL), and MD5, which is described in RFC1321 of the Internet Engineering Task Force, April, 1992.

After configuring SNSL, SNlogin invokes an operating system call, SETVIN, to cause the SNlogin script to run in a Supernet context (step 620). In Unix, each process has a data structure known as the "proc structure" that contains the process ID as well as a pointer to a virtual memory description of this process. In accordance with methods and systems consistent with the present invention, the channel IDs indicating the channels over which the process communicates as well as its virtual address for this process are added to this structure. By associating this information with the process, the SNSL layer can enforce that this process runs in a Supernet context. Although methods and systems consistent with the present invention are described as operating in a Unix

environment, one skilled in the art will appreciate that such methods and systems can operate in other environments. After the SNlogin script runs in the Supernet context, the SNlogin script spawns a Unix program, such as a Unix shell or a service deamon (step 622). In this step, the SNlogin script spawns a Unix shell from which programs can be run by the user. All of these programs will thus run in the Supernet context until the user runs the SNlogout script.

Fig. 7 depicts a flow chart of the steps performed when sending a packet from node A. Although the steps of the flow chart are described in a particular order, one skilled in the art will appreciate that these steps may be performed in a different order. Additionally, although the SNSL layer is described as performing both authentication and encryption, this processing is policy driven such that either authentication, encryption, both, or neither may be performed. The first step performed is for the SNSL layer to receive a packet originating from node A via the TCP/UDP layer and the inner IP layer (step 702). The packet contains a source node ID, a destination node ID, and data. The SNSL layer then accesses the VARPDB to obtain the address mapping between the source node ID and the source real address as well as the destination node ID and the destination real address (step 704). If they are not contained in the VARPDB because this is the first time a packet has been sent from this node or sent to this destination, the VARPDB accesses the local VARPD to obtain the mapping. When contacted, the VARPD on the local machine contacts the VARPD that acts as the server for the Supernet to obtain the appropriate address mapping.

After obtaining the address mapping, the SNSL layer determines whether it has been configured to communicate over the appropriate channel for this packet (step 706). This configuration occurs when SNlogin runs, and if the SNSL has not been so configured, processing ends. Otherwise, SNSL obtains the channel key to be used for this channel (step 708). The SNSL

maintains a local cache of keys and an indication of the channel to which each key is associated. Each channel key is time stamped to expire in ten seconds, although this time is configurable by the administrator. If there is a key located in the cache for this channel, SNSL obtains the key. Otherwise, SNSL accesses KMD which then locates the appropriate channel key from the appropriate KMC. After obtaining the key, the SNSL layer encrypts the packet using the appropriate encryption algorithm and the key previously obtained (step 710). When encrypting the packet, the source node ID, the destination node ID, and the data may be encrypted, but the source and destination real addresses are not, so that the real addresses can be used by the public network infrastructure to send the packet to its destination.

After encrypting the packet, the SNSL layer authenticates the sender to verify that it is the bona fide sender and that the packet was not modified in transit (step 712). In this step, the SNSL layer uses the MD5 authentication protocol, although one skilled in the art will appreciate that other authentication protocols may be used. Next, the SNSL layer passes the packet to the IP layer where it is then sent to the destination node in accordance with known techniques associated with the IP protocol (step 714).

Fig. 8 depicts a flow chart of the steps performed by the SNSL layer when it receives a packet. Although the steps of the flow chart are described in a particular order, one skilled in the art will appreciate that these steps may be performed in a different order. Additionally, although the SNSL layer is described as performing both authentication and encryption, this processing is policy driven such that either authentication, encryption, both, or neither may be performed. The first step performed by the SNSL layer is to receive a packet from the network (step 801). This packet contains a real source address and a real destination address that are not encrypted as well as a

source node ID, a destination node ID, and data that are encrypted. Then, it determines whether it has been configured to communicate on this channel to the destination node (step 802). If SNSL has not been so configured, processing ends. Otherwise, the SNSL layer obtains the appropriate key as previously described (step 804). It then decrypts the packet using this key and the appropriate encryption algorithm (step 806). After decrypting the packet, the SNSL layer authenticates the sender and validates the integrity of the packet (step 808), and then it passes the packet to the inner IP layer for delivery to the appropriate node (step 810). Upon receiving the packet, the inner IP layer uses the destination node ID to deliver the packet.

Fig. 9 depicts a flow chart of the steps performed when logging a node out of a Supernet. The first step performed is for the user to run the SNlogout script and to enter a node ID (step 902). Next, the SNlogout script requests a log out from SASD (step 904). Upon receiving this request, SASD removes the mapping for this node from the VARPD that acts as the server for the Supernet (step 906). SASD then informs KMS to cancel the registration of the node, and KMS terminates this KMC (step 908). Lastly, KMS generates a new channel key for the channels on which the node was communicating (step 910) to provide greater security.

Channel-Specific File System Views

By using the NameI function, the GetDir function, and the file system view table, the Supernet may provide channel-specific file system views. To do so, first the Supernet administrator creates a master version of the file system view table, and then when a node joins the Supernet, a copy of this table is sent to the device on which that node resides. The SASD updates the locally stored tables periodically when the administrator modifies the master version of the table.

Fig. 10 depicts one implementation of a channel-specific file system view table consistent with the present invention. As shown in Fig. 10, FSVT 1000 has four columns: the location 1010, indicating the network location (e.g., a UNIX inode) of the file system entity; the Channel ID 1020 indicating an authorized channel, where the nodes communicating over this channel are authorized to access the file system entity; the device type 1025 that indicates the type of device that may access the file system entity; and the entity name 1030 for that file system entity, which indicates the name that the nodes on the channel use to refer to file system entity. FSVT 1000 contains a number of versions of files, each version being appropriate for a different device type (e.g., Intel, SPARC™ architecture, etc.). For example, entry 1040 indicates that a file system entity known as "Spreadsheet" can be accessed by nodes running on channel 1, and this version of spreadsheet is appropriate for Intel devices. The location of this version of spreadsheet is also specified. Additionally, entry 1050 indicates a version of spreadsheet that is appropriate for a SPARC™ architecture device type. Entry 1060, on the other hand, refers to a text file that can be accessed by any device type. OS 558 uses the FSVT 1000 to ensure that a node may access those file system entities to which it has access to and those file system entities that are compatible with the device architecture of the device on which the node is resident.

Specifically, the NameI function of the operating system handles file system manipulation requests. It is incorporated as part of the Unix system calls open, create, and unlink. Fig. 11 depicts a flow chart of the steps performed when a node makes a file system manipulation request, such as through an open, create, or unlink system call. The first step performed is that the NameI function receives a request indicating an entity name from either the open, create, or unlink Unix system calls (step 1102). Upon receiving this request, the NameI function accesses the file system view table to

retrieve the location of the file system entity (step 1104). In this step, the NameI function receives a name of a file system entity, such as a directory name or file name, and attempts to map it to a network location. First, the NameI function identifies the channel over which the node communicates by accessing the proc structure for the node that made the request. Then, the NameI function identifies the type of the device on which it is running. This information is hard coded into the NameI function. Given the device type and channel ID, the NameI function determines if there is a matching entry in the FSVT (step 1106). If no matching entry is found, then either there was no suitable version of the file for this device type, or the node does not communicate over a channel that is authorized to access this file system entity and so, the NameI function returns an error (step 1107). If an entry is found, the NameI function returns the location of the file system entity to the system call that called NameI (step 1108). For example, if the system call were an open call, it would receive the location of the file and then the caller would be able to open the file.

The above described system also provides channel specific directory listings, in that a node on a particular channel may view those file system entities that have a channel ID corresponding to the channels on which the node may communicate. Fig. 12 depicts a flow chart of the steps performed when a node invokes the GetDir function to view all file system entities that the node is entitled to view. Upon receiving a request, the GetDir function of the operating system accesses the file system view table (step 1202). Then, the GetDir function identifies all files within a node's view (step 1204). In this step, the GetDir function obtains the channel ID of the channel over which the node communicates by accessing the proc structure for the node that made the request. Also, the GetDir function obtains the device type which is hard-coded in the function. Given the channel ID and the device type, the GetDir function accesses the file system view table to identify all of the file

system entities that the node can view and creates a list of these entities. Next, the GetDir function returns the list of file system entities to the user (step 1206).

Finally, the above described system and method may also be used to provide node specific configuration files. This is because the NameI function and the FSVT may be used to map configuration files for each node such that each node has its own specific configuration file referred to by the same entity name. For example, even though there may be many versions of the configuration file at different locations, each node views the files using the same name.

Although the present invention has been described with reference to a preferred embodiment thereof, those skilled in the art will know of various changes in form and detail which may be made without departing from the spirit and scope of the present invention as defined in the appended claims and their full scope of equivalents.

WHAT IS CLAIMED IS:

1. A distributed system with a network having devices with nodes communicating over a first channel and nodes communicating over a second channel, one of the devices comprising:

a secondary storage device comprising:

a plurality of file system entities, a first of the file system entities accessible by the nodes communicating over the first channel and a second of the file system entities accessible by the nodes communicating over the second channel;

a memory comprising:

an operating system that restricts access to the first file system entity to the nodes communicating over the first channel and that restricts access to the second file system entity to the nodes communicating over the second channel; and

one of the nodes that communicates over the first channel that sends a request to access the first file system entity to the operating system; and

a processor for running the operating system and for running the one node.

2. The distributed system of claim 1 wherein the network is a private network running over a public network infrastructure.

3. The distributed system of claim 1 wherein the first file system entity is a file, wherein the devices have a plurality of types, wherein the secondary storage device includes a plurality of versions of the file, each version suitable to a type, and wherein when the operating system receives the request from the one node, the operating system returns the version of the file that is suitable for the type of the one device.

4. The distributed system of claim 1 wherein the operating system maintains a mapping between each of the plurality of file system entities and an authorized channel such that nodes communicating over the authorized channel are authorized to access the associated file system entity.

5. The distributed system of claim 4 wherein the operating system includes a function that receives a request from the one node, that determines that the one node communicates over the first channel, and that returns a list of file system entities where the authorized channel is the first channel.

6. The distributed system of claim 5 wherein the one node is a process with an associated proc structure that indicates that the one node communicates over the first channel and wherein the function determines that the one node communicates over the first channel by accessing the proc structure.

7. The distributed system of claim 1 wherein the operating system includes an open function that receives a request from the one node to open the first file system entity, that verifies that the one node communicates over the one channel, and that opens the first file system entity when the open function verifies that the one node communicates over the first channel.

8. The distributed system of claim 1 wherein the operating system includes an unlink function that receives a request from the one node to unlink the first file system entity, that verifies that the one node communicates over the one channel, and that deletes the one file system entity when the unlink function verifies that the one node communicates over the first channel.

9. A method in a distributed system with a network of nodes communicating over channels, comprising the steps of:

receiving a request from one of the nodes to access a file system entity, the file system entity having an associated authorized one of the channels;

determining whether the node communicates over the authorized channel; and

accessing the file system entity when it has been determined that the node communicates over the authorized channel.

10. The method of claim 9, wherein the method further includes the steps of:

inhibiting access to the file system entity when it has been determined that the node does not communicate over the authorized channel.

11. The method of claim 9 wherein the file system entity is a file, wherein the network has a plurality of devices, each with an associated type, wherein the method is performed by one of the devices, and wherein the network stores a plurality of versions of the file, one version for each type and wherein the accessing step further includes the steps of:

identifying the version of the file that is suitable for the type of the one device; and

accessing the identified version.

12. The method of claim 9, wherein the method is performed by an operating system.

13. A distributed system with a network of nodes communicating over channels, comprising:

means for receiving a request from one of the nodes to access a file system entity, the file system entity having an associated authorized one of the channels;

means for determining whether the node communicates over the authorized channel; and

means for accessing the file system entity when it has been determined that the node communicates over the authorized channel.

14. A method in a distributed system with a network having nodes communicating over channels, the network having a file system with a plurality of file system entities, each with an associated authorized channel, the method comprising the steps of:

receiving a request from one of the nodes communicating over one of the channels, the request for viewing the file system;

determining which file system entities in the file system have an authorized channel as the one channel; and

returning an indication of the determined file system entities.

15. A computer-readable memory device encoded with a data structure for use by an operating system in providing channel-specific views of a file system, the data structure having entries, each entry comprising:

an indication of a file system entity; and

an indication of a channel that is used by the operating system to restrict access to the file system entity to nodes that communicate over the indicated channel.

16. The computer-readable memory device of claim 15, wherein the indication of a file system entity is an indication of a version of a file and wherein each entry further includes:

an indication of a device type that is used by the operating system to determine whether a requesting node that requests access to the version of the file runs on a device having the indicated device type.

17. A computer-readable medium containing instructions for controlling a distributed system with a network of nodes communicating over channels to perform a method comprising the steps of:

receiving a request from one of the nodes to access a file system entity, the file system entity having an associated authorized one of the channels;

determining whether the node communicates over the authorized channel; and

accessing the file system entity when it has been determined that the node communicates over the authorized channel.

18. The computer-readable medium of claim 17, wherein the method further includes the steps of:

inhibiting access to the file system entity when it has been determined that the node does not communicate over the authorized channel.

19. The computer-readable medium of claim 17, wherein the file system entity is a file, wherein the network has a plurality of devices, each with an associated type, wherein the method is performed by one of the devices, and wherein the network stores a plurality of versions of the file, one version for each type and wherein the accessing step further includes the steps of:

identifying the version of the file that is suitable for the type of the one device; and

accessing the identified version.

20. The computer-readable medium of claim 17, wherein the method is performed by an operating system.

ABSTRACT

Methods and systems consistent with the present invention provide a Supernet, a private network constructed out of components from a public-network infrastructure. The Supernet provides channel-specific file system views such that the file system of the Supernet is partitioned on a per-channel basis so that nodes on one channel see a different view of the network file system than the nodes on a different channel.

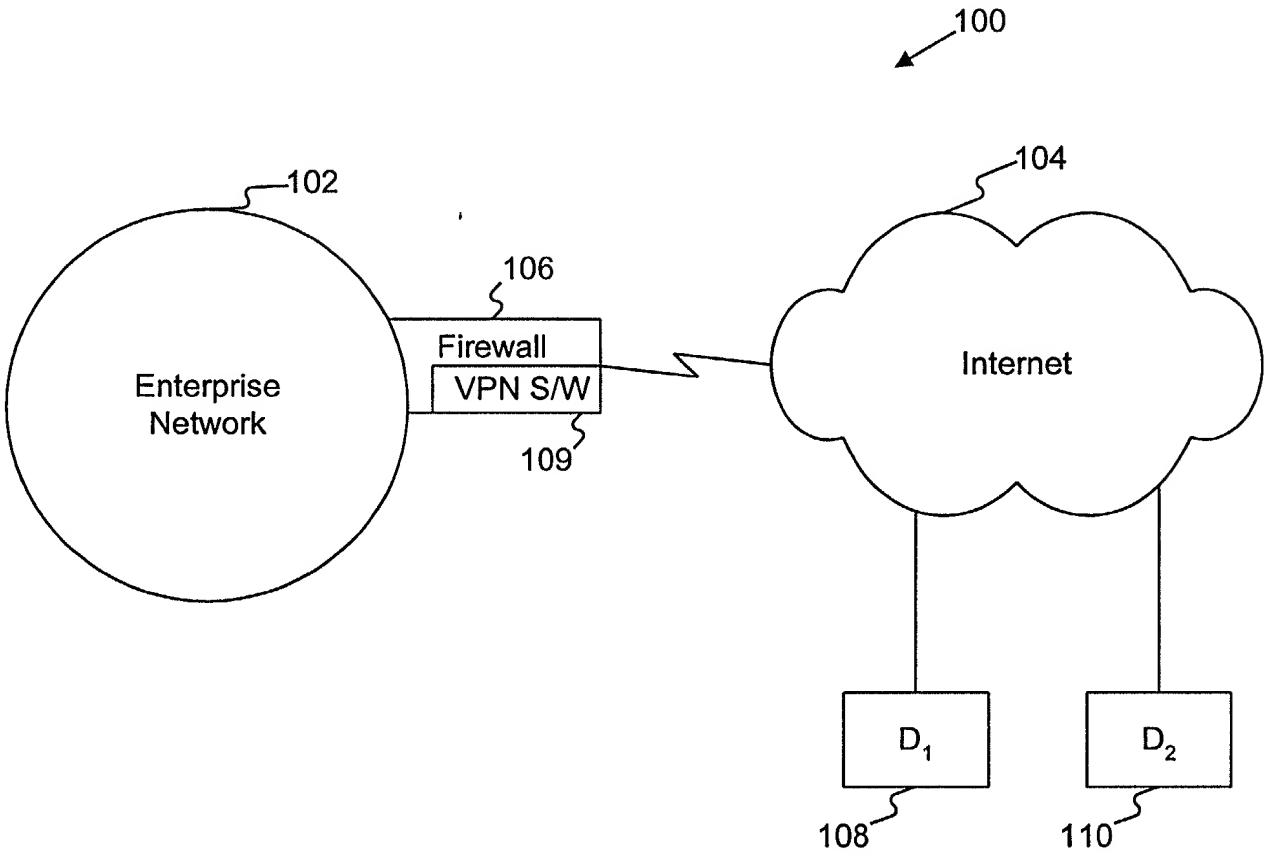

Fig. 1
(Prior Art)

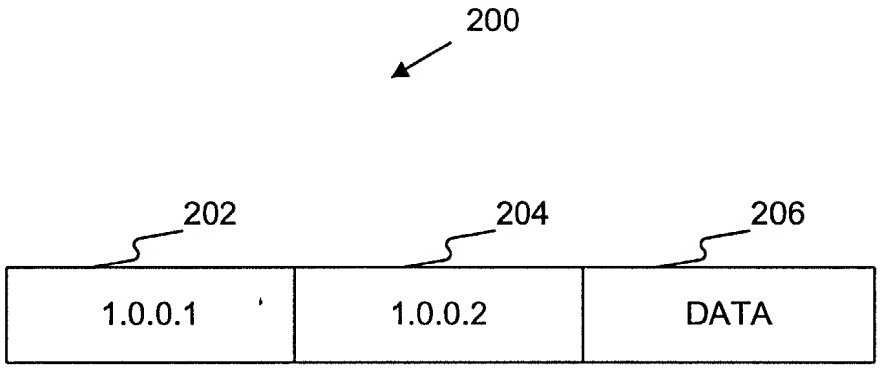

Fig. 2A
(Prior Art)

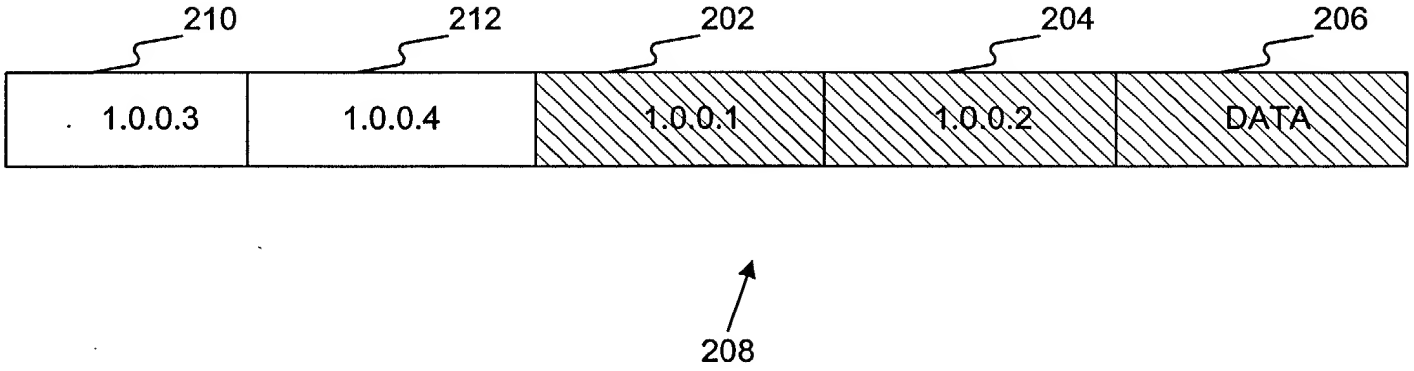

Fig. 2B
(Prior Art)

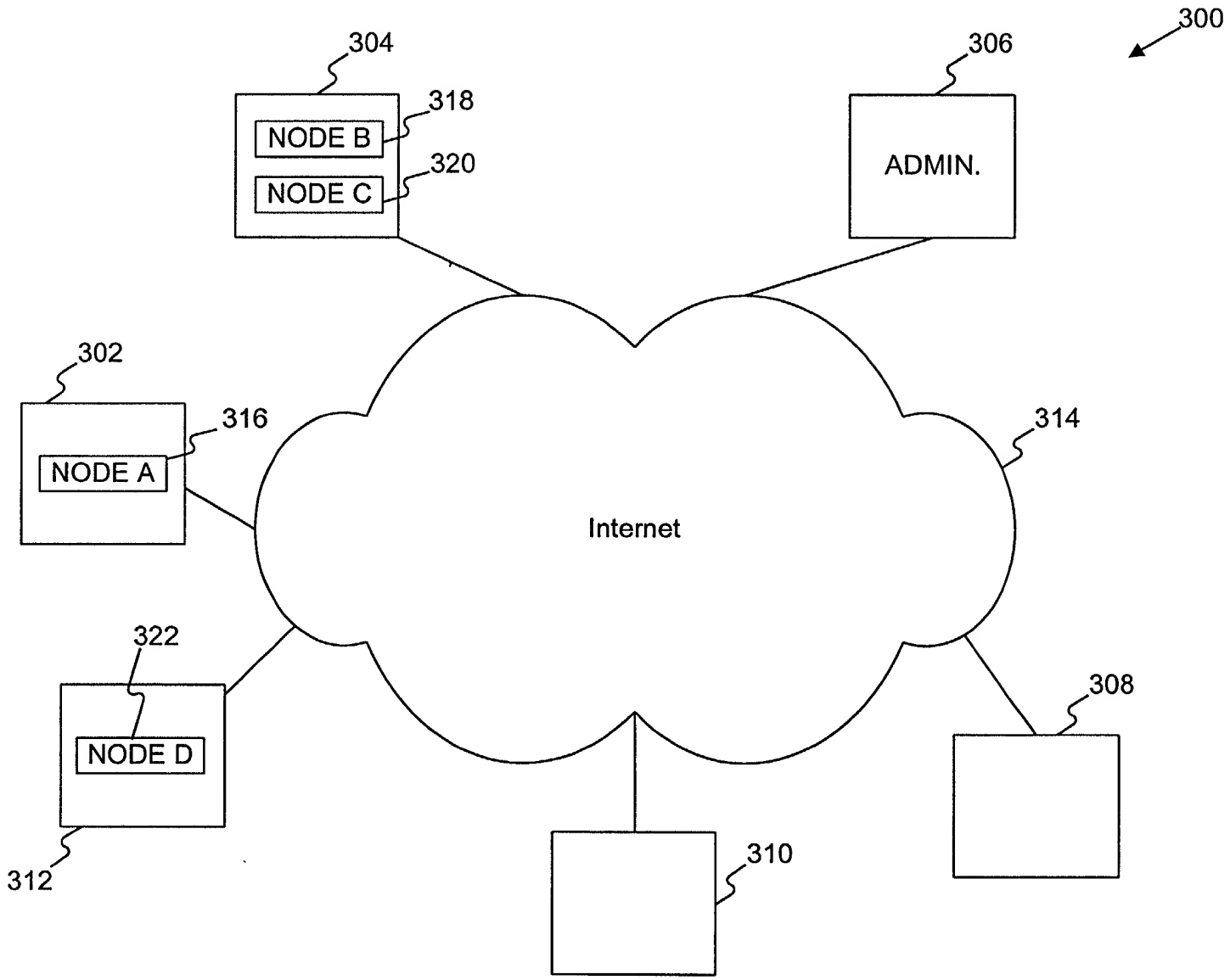

Fig. 3

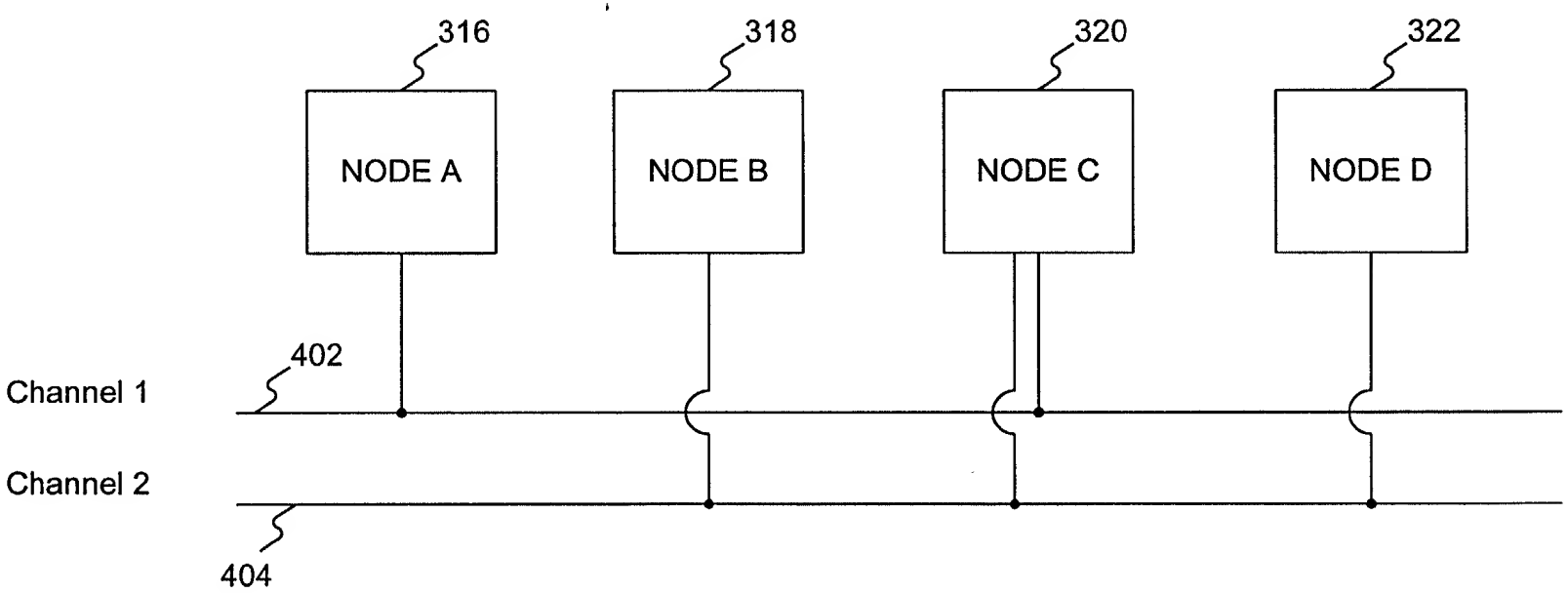

Fig. 4

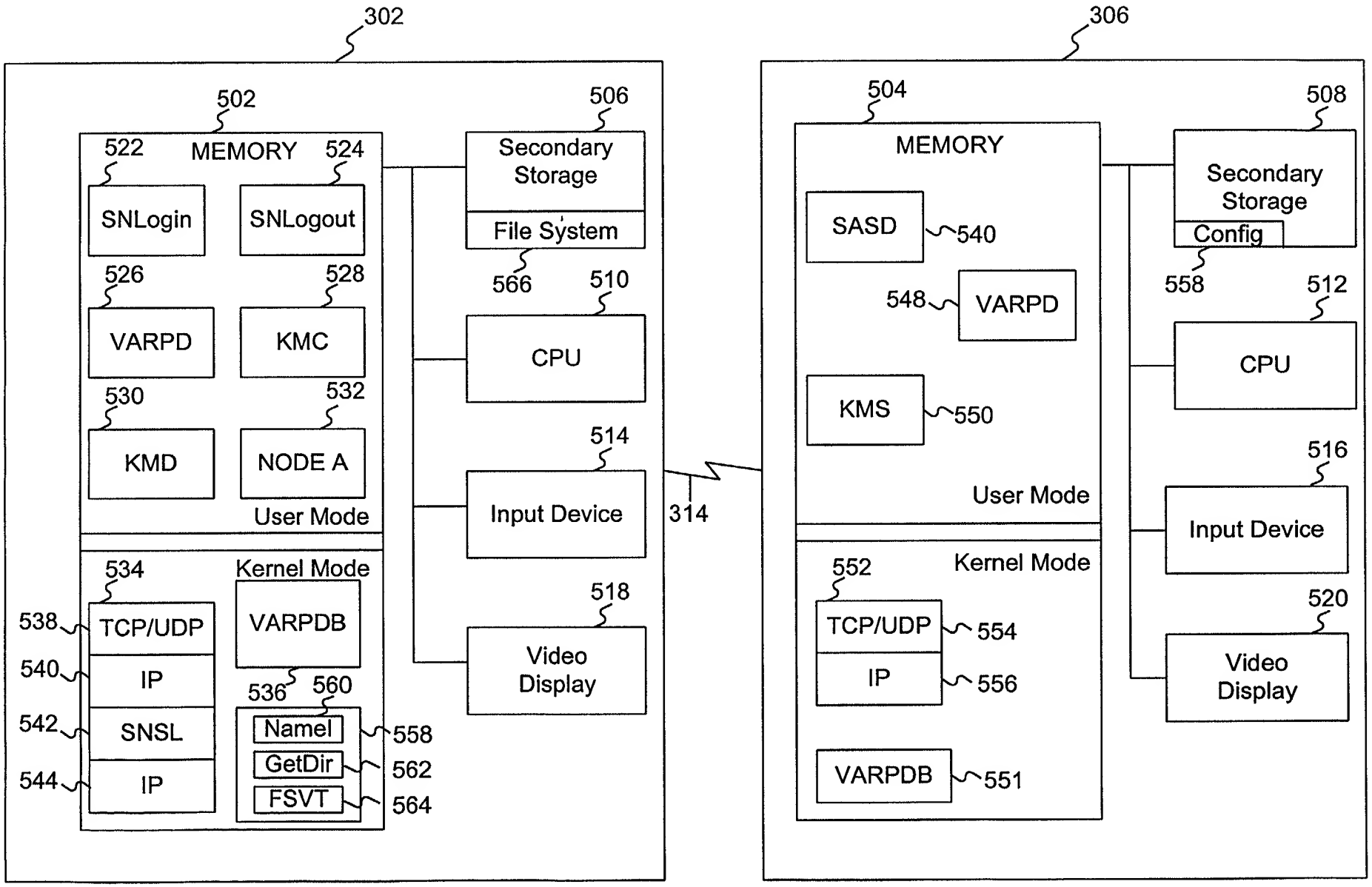
302
306
502
504
506
508
522
MEMORY
524
MEMORY
SNLogin
SNLogout
SASD
540
Secondary Storage
File System
Secondary Storage
Config
526
528
548
VARPD
566
510
558
512
VARPD
KMC
CPU
CPU
530
532
KMS
550
514
516
KMD
NODE A
User Mode
Input Device
User Mode
Input Device
314
534
Kernel Mode
552
Kernel Mode
518
520
538
TCP/UDP
VARPDB
TCP/UDP
554
Video Display
Video Display
540
IP
536
560
IP
556
542
SNSL
Namel
558
544
IP
GetDir
562
FSVT
564
VARPDB
551

Fig. 5

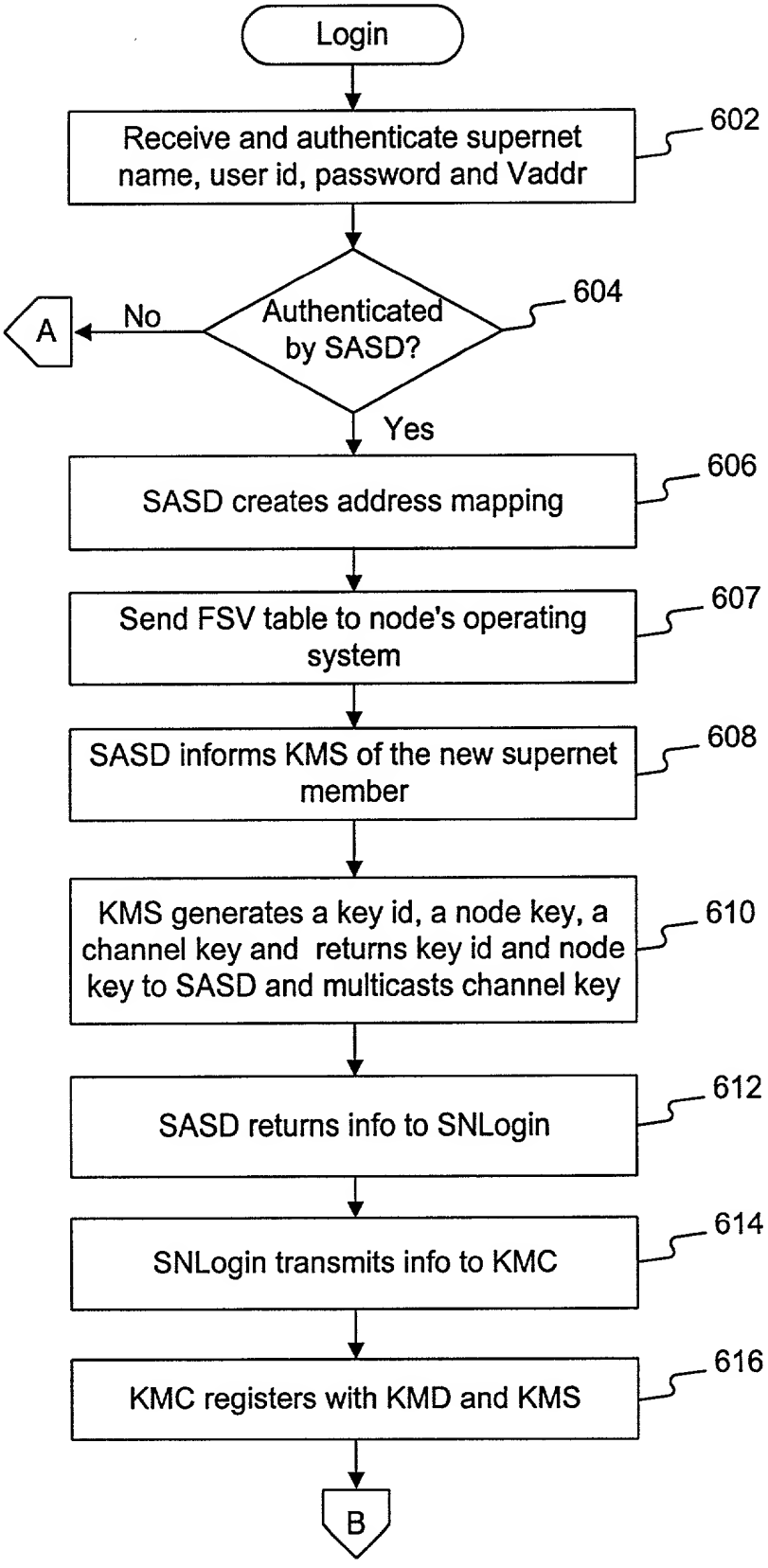
Login
Receive and authenticate supernet name, user id, password and Vaddr
602
Authenticated by SASD?
604
No
A
Yes
SASD creates address mapping
606
Send FSV table to node's operating system
607
SASD informs KMS of the new supernet member
608
KMS generates a key id, a node key, a channel key and returns key id and node key to SASD and multicasts channel key
610
SASD returns info to SNLogin
612
SNLogin transmits info to KMC
614
KMC registers with KMD and KMS
616
B

Fig. 6A

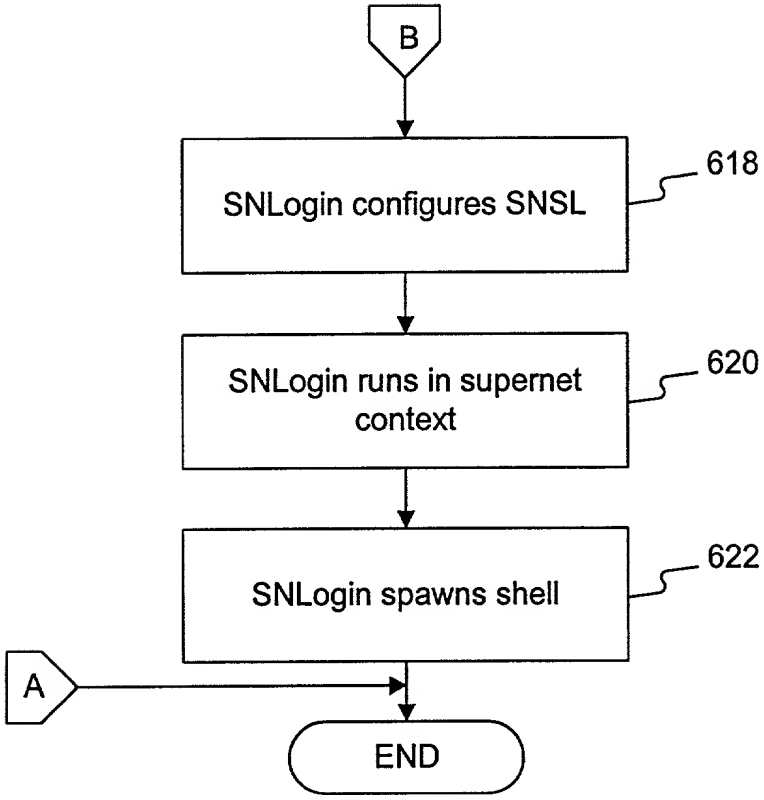
B
SNLogin configures SNSL
618
SNLogin runs in supernet context
620
SNLogin spawns shell
622
A
END

Fig. 6B

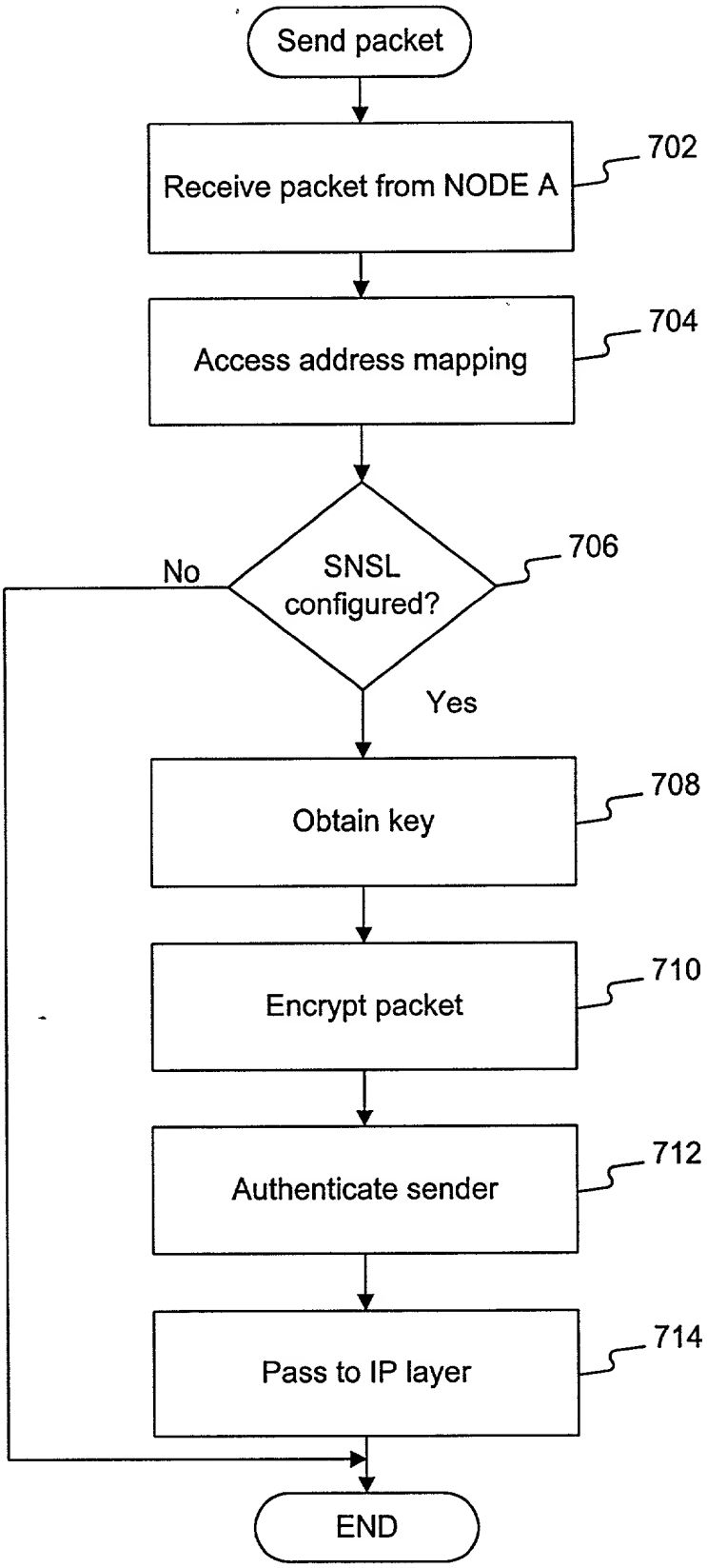
Send packet
Receive packet from NODE A
702
Access address mapping
704
SNSL configured?
706
No
Yes
Obtain key
708
Encrypt packet
710
Authenticate sender
712
Pass to IP layer
714
END

Fig. 7

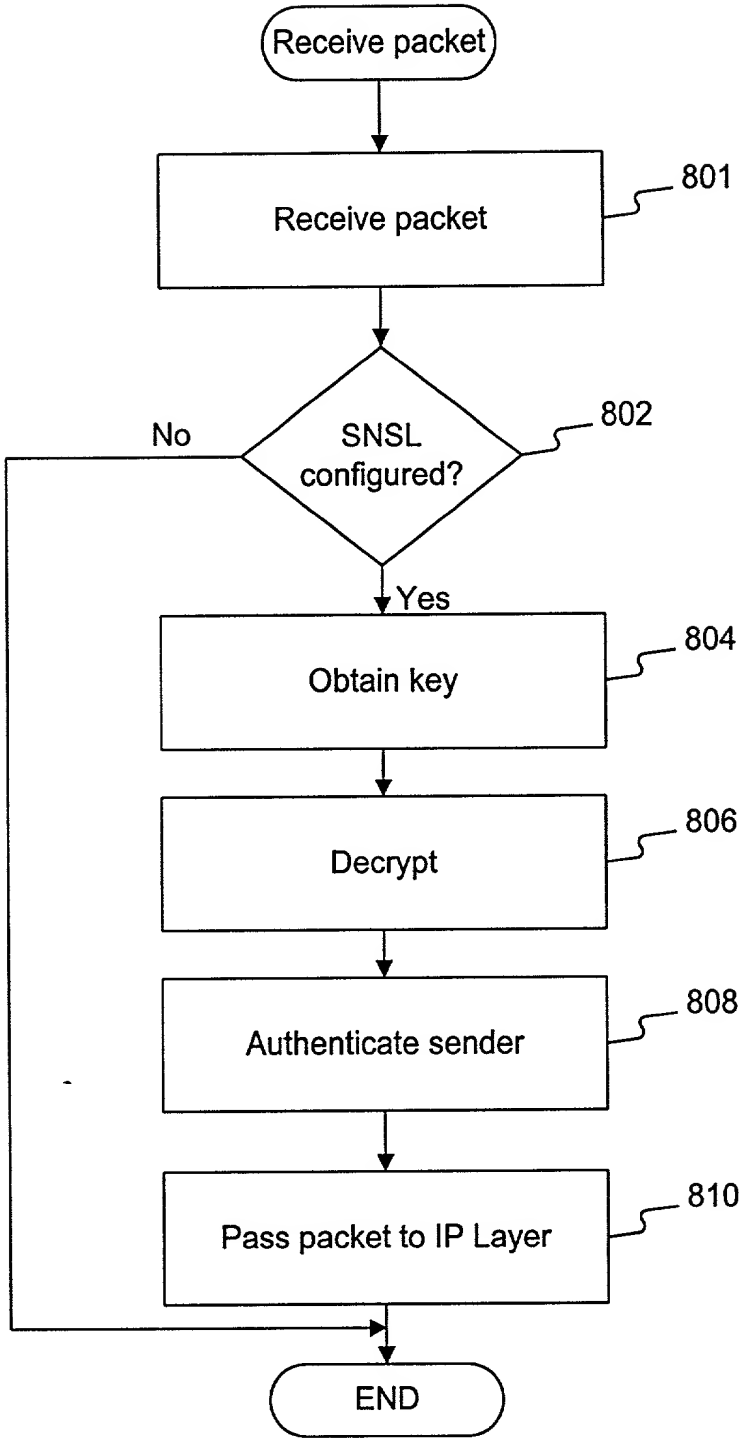
Receive packet
Receive packet
801
SNSL configured?
802
No
Yes
Obtain key
804
Decrypt
806
Authenticate sender
808
Pass packet to IP Layer
810
END

Fig. 8

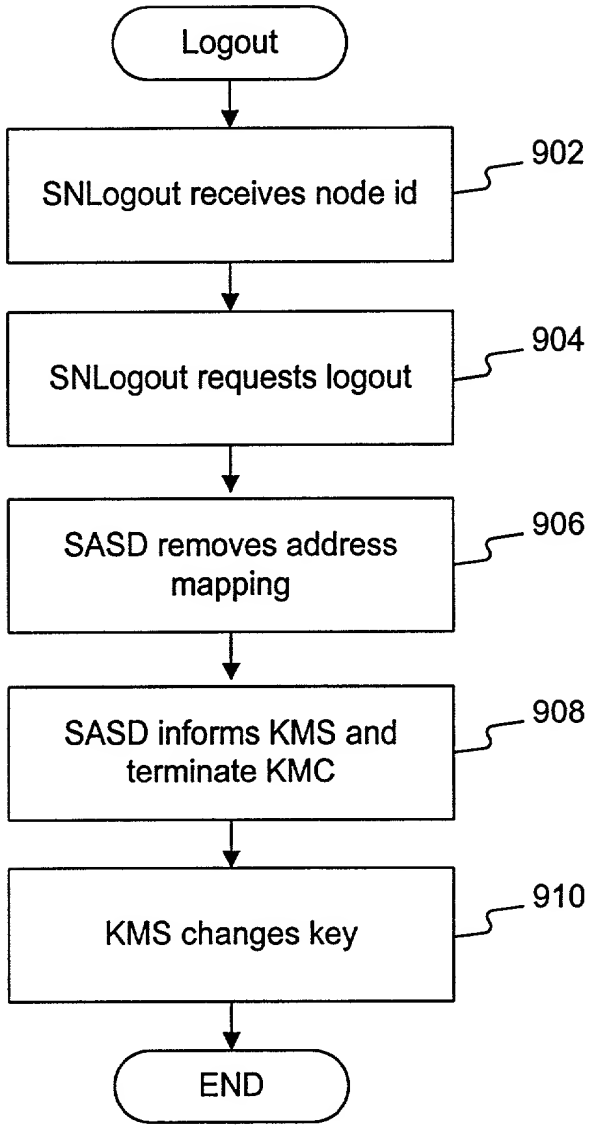
Logout
SNLogout receives node id
902
SNLogout requests logout
904
SASD removes address mapping
906
SASD informs KMS and terminate KMC
908
KMS changes key
910
END

Fig. 9

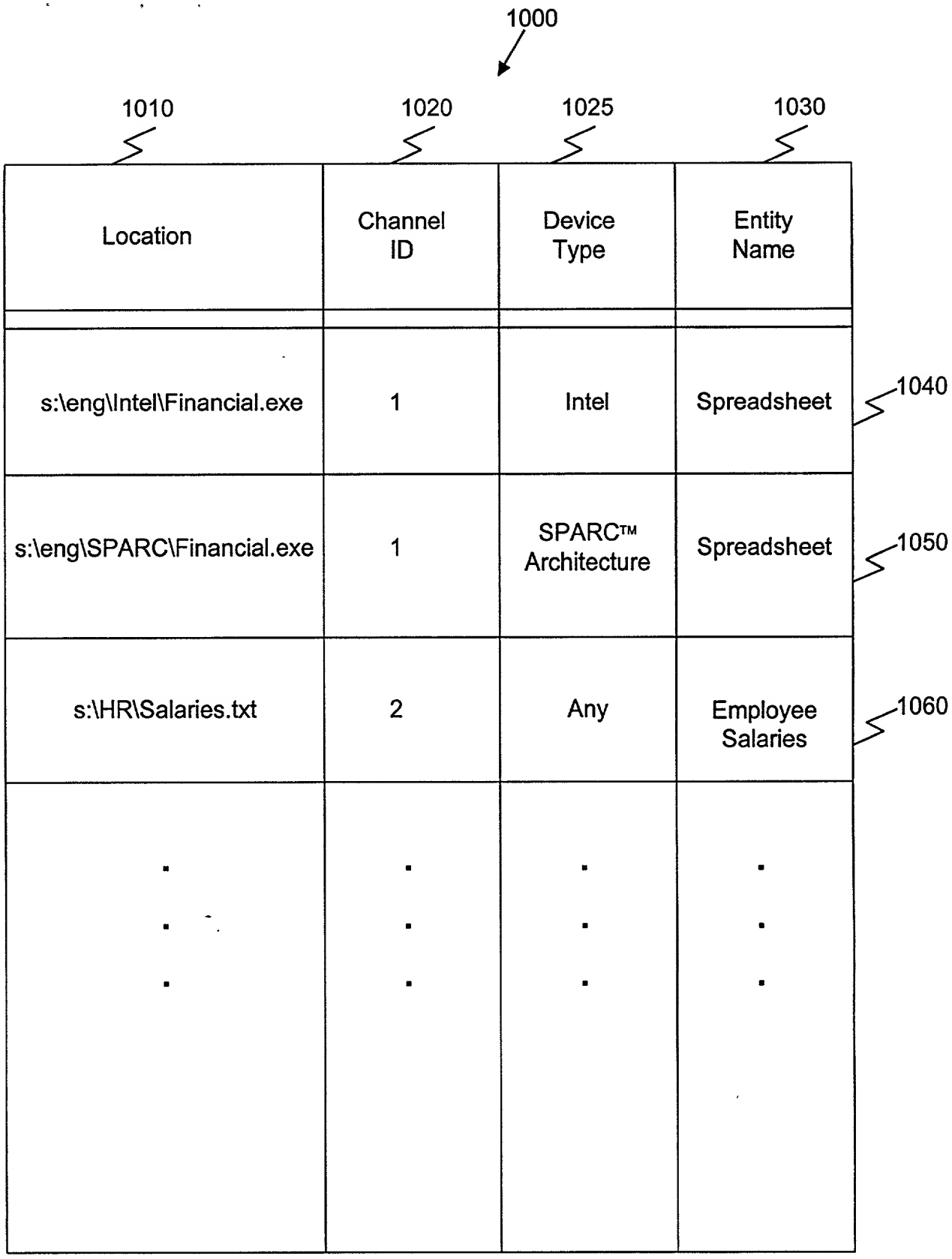
1000
1010
1020
1025
1030
Location
Channel ID
Device Type
Entity Name
s:\eng\Intel\Financial.exe
1
Intel
Spreadsheet
1040
s:\eng\SPARC\Financial.exe
1
SPARC™ Architecture
Spreadsheet
1050
s:\HR\Salaries.txt
2
Any
Employee Salaries
1060

Fig. 10

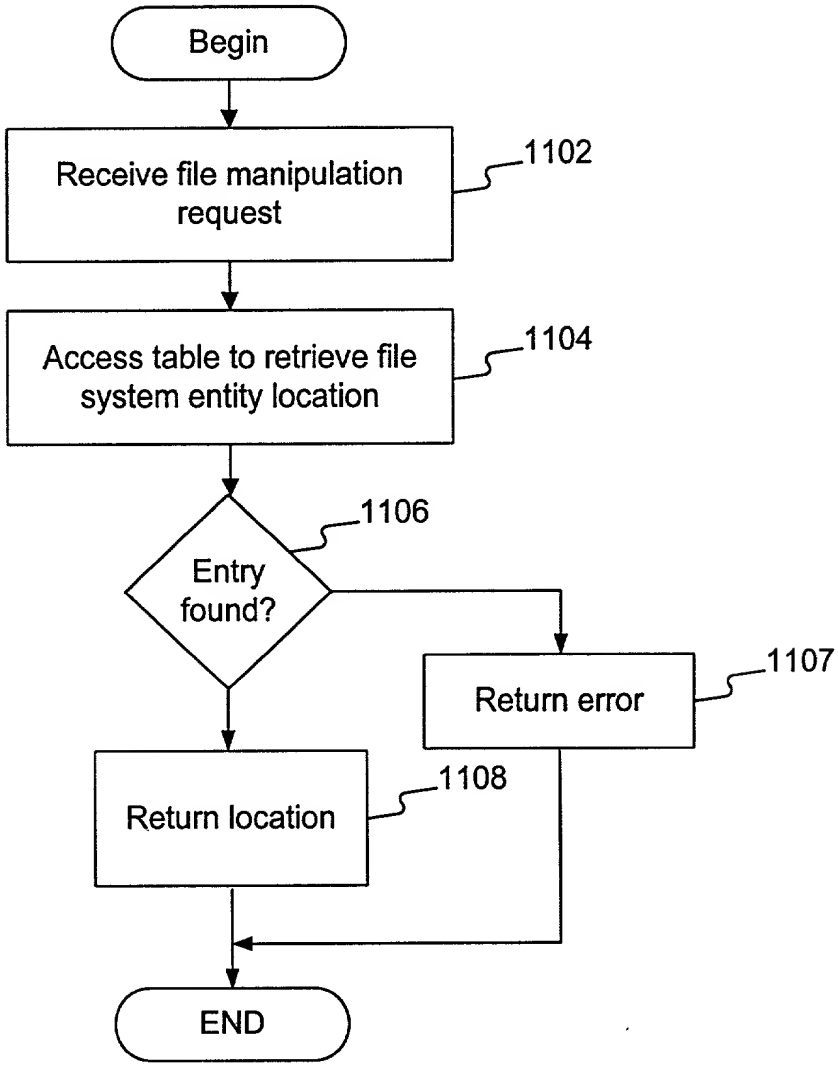
Begin
Receive file manipulation request
1102
Access table to retrieve file system entity location
1104
Entry found?
1106
Return error
1107
Return location
1108
END

Fig. 11

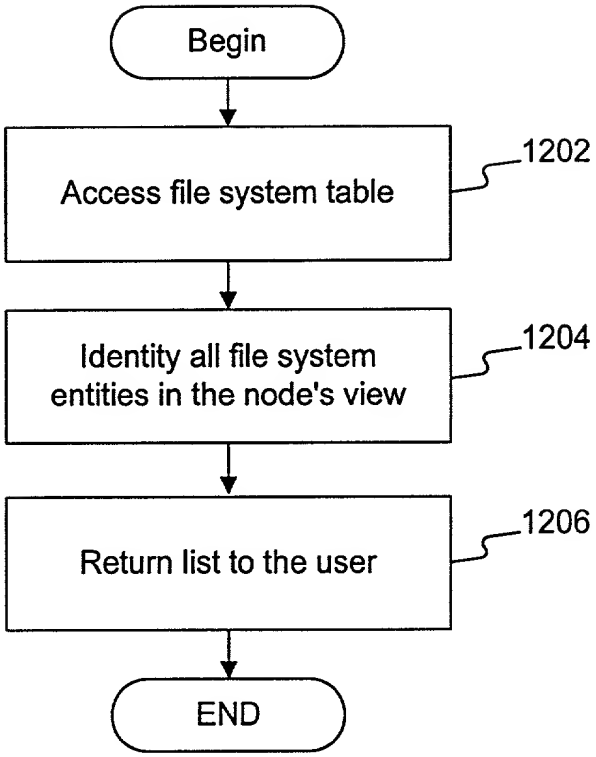
Begin
Access file system table
1202
Identity all file system entities in the node's view
1204
Return list to the user
1206
END

Fig. 12